Int. J. Inf. Secur.
DOI 10.1007/s10207-016-0323-3

REGULAR CONTRIBUTION

A key management scheme evaluation using Markov processes

Hani Ragab-Hassen[1] · **Esma Lounes**[1]

Abstract Content access control aims at ensuring that, in a system with several resources, users can only access resources they are authorized to. Resources are encrypted using cryptographic keys. Generating, distributing and renewing these keys are the challenges faced by key management schemes. While most of the existing key management schemes are typically evaluated by simulation. We propose, for the first time, to use Markovian processes for this purpose. Markovian processes give more accurate evaluation. The key tables-based key management scheme for linear hierarchies (KTLH) is a particularly interesting key management scheme; it was initially proposed for securing group communications, but could easily be adapted to other application such as wireless sensor networks. KTLH requires each user to maintain a set of keys. The keys and size of the key set change dynamically, making the evaluation of the overheads of KTLH a challenging task. Our contribution is threefold, we have (1) modeled KTLH using Markov processes, (2) evaluated KTLH according to its storage, computation and bandwidth overheads and compared it to existing key management schemes and (3) shown how our approach could be generalized to other key management schemes.

Keywords Key management · Markov processes · Content access control · Linear hierarchies

✉ Hani Ragab-Hassen
h.ragabhassen@hw.ac.uk

Esma Lounes
el10@hw.ac.uk

[1] School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK

1 Introduction

Ensuring content access control is a critical security issue within hierarchies. A hierarchy is defined using a set of access rights (privileges), and each entity in the hierarchy has a subset of these access rights. Content access control in a hierarchy (CACH) is required in any context where a set of sensitive information items should be accessed with a set of access rights. A user might have access to particular items while others do not. The main goal of CACH is to ensure that users access only items to which they are entitled. CACH is required in several contexts such as ensuring access control to resources shared on a cloud or protecting messages in a hierarchical group (e.g., military communications).

For example, in a multilayered video streaming application with a basic layer quality, and N enhancement layers (EL). A user who pays for enhancement layer EL_i should be able to access enhancement layers of lower quality, but not to enhancement layers of higher quality. Ensuring content access control in this example can be carried out by encrypting the different enhancement layers using different keys. Each user will have the key of the enhancement layer they are entitled to, as well as (the required information to get) the keys for the enhancement layers of lower quality. These keys are called Resource Encryption Keys (REK). Key management for CACH aims at efficiently generating, distributing and renewing these keys while fulfilling the security requirements of content access control [1].

Key management for CACH is a challenging security problem. The key management problem has already been addressed for flat systems. In a flat system, there is only one data stream. That is, a user can be authorized to access either all data or nothing. Adding a hierarchy makes the key management more complex. Indeed, in a hierarchy, there are several data streams, each encrypted using a different keys.

Published online: 04 June 2016

Markov processes (aka continuous time Markov chain *CTMC*) are a category of stochastic processes that have the Markov property [2]. They have been extensively studied in the stochastic processes literature [3–5], and interesting properties have been established for them. Numerical properties of key management schemes can be modeled by Markov processes.

1.1 Our contribution

In this paper, we show how to model key management schemes using Markov models. We have chosen the *key tables-based key management scheme for linear hierarchies* (KTLH) key management scheme [6] because of its complex and random behavior. Applying Markov processes to other key management schemes is likely to be of similar or less complexity. Our contribution is threefold, we have (1) modeled KTLH using Markov processes, (2) evaluated KTLH according to its storage, computation and bandwidth overheads and compared it to existing key management schemes and (3) shown how our approach could be generalized to other key management schemes.

The next section reviews the problem of content access control in hierarchies and its key management issues, and gives an overview of the existing key management schemes including KTLH. Our Markov model and its results are presented in Sect. 3. In Sect. 4, we evaluate the overheads of KTLH using the results of our model. We present our conclusions in Sect. 5. An appendix contains the required theoretical background on Markov processes.

2 Related works

2.1 Content access control in hierarchies

A hierarchy can be defined in two steps. First, the entities within the system are divided into *security classes (SC)* according to their access rights. Second, an ordering relation is defined on the classes. Ordering the classes is based on the access rights. These access rights allow to say that a class is more privileged than another, thereby defining the hierarchy. A security class contains users who have exactly the same access rights. Users could be individuals, devices or processes.

A security class SC_i covers (or dominates) another security class SC_j and we denote it by $SC_i \succeq SC_j$ if and only if any user in SC_i has access to all information items in SC_j. Users within the same security class could be assigned the same role in a *role-based access control (RBAC)* context.

A very interesting type of hierarchy is the linear hierarchies. In such a hierarchy, security classes form a directed chain. The first class in the chain is the most privileged.

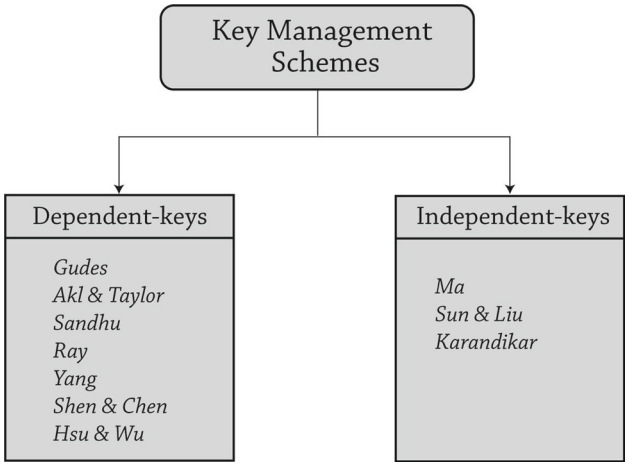

Fig. 1 A classification of CACH key management schemes

Indexes can be assigned such that $i < j \Leftrightarrow SC_i \succ SC_j$. That is, SC_1 is the highest class.

2.2 Key management schemes for CACH

Key management typical tasks are Resource Encryption Keys (REK) generation and renewal [1]. The REK renewal is necessary to ensure confidentiality when users join or leave the hierarchy. That is, a new user must be prevented from accessing data exchanged before they join the hierarchy. This requirement is called *backward secrecy*. Inversely, a leaving user should not have access to data exchanged after they leave the session (*forward secrecy* requirement). A renewal must be carried out after their departure.

The existing key management schemes for CACH can be divided into two approaches: the *dependent-keys* and the *independent-keys* approaches [1] as shown in Fig. 1.

The independent-keys approach originates from the multicast security community. More precisely, from works on key management, hence, these schemes use usual key trees and graphs techniques [7]. In order to access any information item, the user must have a copy of its decryption key.

In the dependent-keys approach, in order to access a given information item (message), a legitimate user does not need to have the key with which it is encrypted. But using merely their own key, combined with some public parameters and/or functions, they can compute the key used to decrypt the item. The resulting key management schemes typically use complex cryptographic techniques.

The dependent-keys schemes can be further divided in two categories as shown in Fig. 2: *indirect access schemes* and *direct access schemes*. In indirect access schemes, if $SC_i \succ SC_j$, members of SC_i must compute all intermediate keys on the path from SC_i to SC_j in order to be able to compute the key of SC_j. Sandhu [8,9], Gudes [10] and Yang and Li [11] proposed indirect schemes that are based on one-way

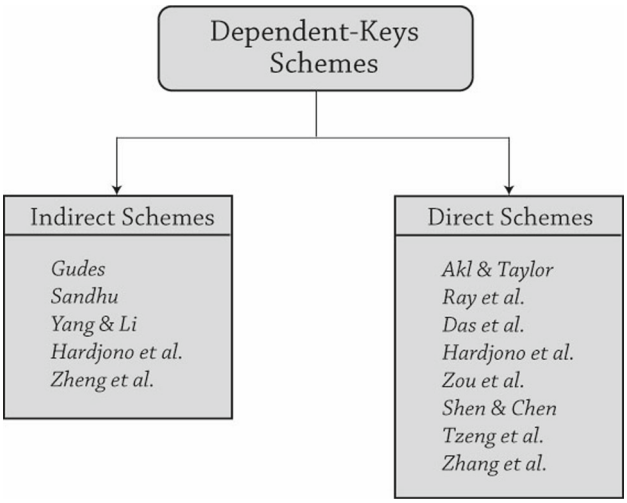

Fig. 2 Indirect and direct key management schemes

functions. In these schemes, the key of a class is computed from the key of its parent using a one-way function.

A direct scheme avoids the computation of intermediate keys by using additional public parameters. Most of the schemes proposed in this approach use prime numbers fundamental properties (Akl and Taylor [12], Ray et al. [13], Zou et al. [14]) and other theoretical cryptographic notions (Hardjono et al. [15] and Zheng et al.[16]). Shen and Chen [17], Zhang et al. [18], Tzang et al. [19] and Das et al. [20] used Newton's polynomial interpolation to correlate classes' indices with their keys.

2.2.1 Discussion

Independent-keys schemes are quite simple to deploy. However, they do not offer an efficient support for hierarchy changes. On the other hand, dependent-keys schemes have the advantage of minimizing the number of keys maintained by users. This is carried out using well-chosen public parameters. However, in most of the schemes proposed in this approach, the number of public parameters is quite important which obsoletes the idea of reducing storage on user's side. One-way functions-based techniques is a good solution for storage overhead issues. In these techniques, keys of the security classes are iteratively generated using a one-way function (possibly combined one or more public parameters). Linking the keys using such functions allows replacing a set of keys by only one key. Nevertheless, changing one key in a set of related keys implies the renewal of the whole set. This generates an additional renewal overhead. KTLH is based on one-way functions but uses a *keys table* mechanism to avoid renewing the whole set of keys. It is important to understand how KTLH works before we describe our model. We will briefly recall it in the next section.

2.3 KTLH

KTLH is a dependent-keys key management scheme for linear hierarchies. KTLH allows users to join and leave the hierarchy, as well as to be promoted (moving to a higher class) or degraded (moving to a lower class) within the hierarchy.

KTLH links keys in such a way that, knowing their own keys table, a user can compute keys of lower classes. Keys tables are used in KTLH to maintain the links between the keys.

Initially, KTLH generates a random key K_1 and uses a hash function H to compute a chain of keys using the formula: $K_{c+1} = H(K_c)$, where K_c is the key of the c^{th} class. Then each key K_c is sent to its corresponding class SC_c. Thus, only one key is sent to each user. Once a user in class SC_c receives their key, they can, if required, compute the key of any class SC_u, $SC_u < SC_c$, by simply applying $(u-c)$ times H to K_c. Since the keys are renewed several times, we denote by K_c^p the key of SC_c after the $(p-1)^{th}$ key renewal. That is, p is the version of the key. The initial key of SC_c is denoted K_c^1.

The novelty in KTLH is the use of *keys tables* to maintain the key chain. Initially, the table of a user in SC_c only contains the pair (c, K_c^1), where K_c^1 is the initial key of SC_c. The keys tables evolve with the hierarchy membership changes. For instance, when a user leaves the c^{th} class, the key K_c^1 of SC_c and keys of all lower (but not higher) classes are renewed (because the leaving user knows them). An update message, containing the pair (c, K_c^2), is sent to higher classes who update their keys tables by adding an entry to it to contains the pair. So, for example, users in class 1 will have in their keys tables, in addition to the pair $(1, K_1^2)$, the pair (c, K_c^2). Note that the key of SC_1 remains unchanged. This is why its upper index (version) remains the same.

Obviously, if the update messages are not used, users in SC_1 will not be able to find K_c^2. This is because it was not computed based on K_{c-1}^1 using H. In order to overcome chain discontinuity problem, users maintain keys tables. Each time that a user receives an update message, they make necessary updates to their keys tables. Later on, when they need to compute the key of a lower class u, the user looks in their table for the greatest class index c such that $c \leq u$. KTLH is a dependent-keys key management scheme because it uses a one-way function. The next examples illustrates how KTLH builds keys tables.

Example Let us consider a system with five security classes $(C = 5)$. Initially, every user needs to know only her own class key. Assume that a user in SC_2 has just been demoted to SC_4. In order to prevent this user from accessing future communications/information items of SC_2 and SC_3, K_2^1 and K_3^1 must be renewed. Note that K_4^1 does not need to be changed since the user had access to it before moving to SC_2 and will keep that access after moving. KTLH generates a new

Table 1 Keys table maintained by SC_1 users

Security class	Key
1	K_1^1
2	K_2^2
4	K_4^1

Table 2 Keys table maintained by SC_3 users

Security class	Key
3	K_3^2
4	K_4^1

key K_2^2 for SC_2. Then, it computes K_3^2 by: $K_3^2 = H(K_2^2)$. That is, the new content encryption keys are K_2^2 and K_3^2, whereas K_1^1, K_4^1 and K_5^1 remain unchanged. KTLH sends K_2^2 to users in SC_2 and K_3^2 to those in SC_3. It also sends keys tables update messages to SC_1, SC_2 and SC_3; users of SC_2 and SC_3 will receive only one message containing the pair $(K_4^2, 4)$, whereas users of SC_1 will receive two messages: the first containing $(K_2^2, 2)$ and the second containing $(K_4^2, 4)$.

Upon receiving these messages, users of SC_1 will update their keys tables to look like Table 1, while users of SC_3 will get Table 2.

3 A Markov process model for KTLH

We will evaluate KTLH according to the following metrics:

(1) *Key Storage* the average number of keys that each user has to store within their keys table;
(2) *Computation Time* the average number of computations of the one-way function that a user should carry out to compute the key of a lower class.
(3) *Bandwidth* number of messages sent per user in order to make key renewals when a membership change occurs.

We propose to model KTLH using a Markov process in order to evaluate these performance criteria. A Markov process is a stochastic process whose future state, given the present state does not depend on the past. That is, the conditional probability distribution of the process depends only on the present state [21]. Appendix 1 contains a summary of the Markov processes properties we are using in this paper.

Our objective is to model the system characteristics by a Markov process and then use the limiting distribution of the Markov process to compute the average value of the modeled variable. This value will serve us as a means to compute the storage and computation overheads.

Let us consider a hierarchy with N classes and let $Z_t(c)$ be the number of keys within keys tables of a member of

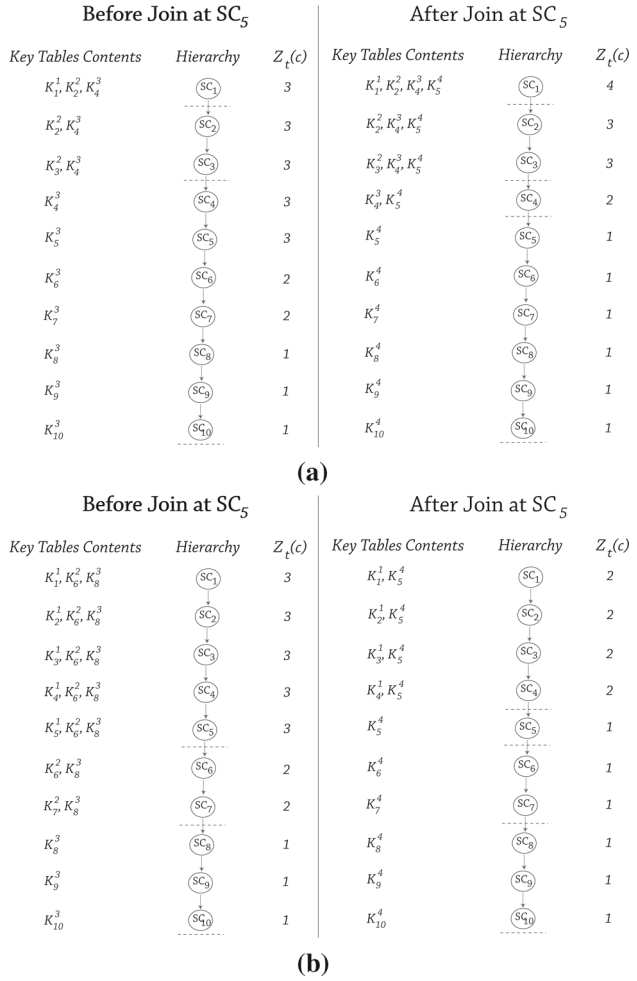

Fig. 3 Keys tables size process

SC_c. Obviously, $Z_t(c)$ is a particularly interesting process because it directly gives the storage overhead, and the average value of $Z_t(c)$ is the average key storage. We thought it is worth justifying to the reader why we decided not to use this process. Indeed, $Z_t(c)$ is not a Markov process because the size of the table of SC_c after a given membership change can have several values independently from its size before the membership change. In other words, the future probability distribution of the table size is not fully determined by knowing its current size. We give a counterexample below to show that $Z_t(c)$ is not a Markov process.

Let $Z_t(1)$ be number of keys within the keys table of SC_1. Suppose that the system contains 10 classes and that the current keys table size is 3. Suppose that a member joins the system at SC_5. If the keys table initially contains K_1^1, K_2^2 and K_4^3 (Fig. 3a), then after the key renewal the keys table will contain K_1^1, K_2^2, K_4^3, K_5^4 (the key of SC_5 is renewed), and hence, the new keys table size is 4. However, if the keys table initially contained K_1^1, K_6^2 and K_8^3 (Fig. 3b), then after the renewal the keys table would have contained K_1^1 and K_5^4

(note that there is no more need to keep K_6^2 and K_8^3 because their new versions can be obtained, thanks to KTLH and the keys table mechanism, from the new key K_5^4). The new keys table size is 2. That is, $Z_t(1)$ has not the same value for both cases after the membership change even if it had the same size, 3, before it. This is due to the fact that $Z_t(1)$ depends on the contents of the keys table, which is related to the process history, thus fulfilling the Markov property. In what follows, we will consider an alternative process which is a Markov one.

3.1 The "Next Discontinuity" process

Let N be the number of security classes. We say that there is a *chain discontinuity* between two successive classes SC_i and SC_{i+1} if K_{i+1} cannot be computed by applying the one-way function to K_i (e.g., due to the previous key renewal triggered by a user leaving SC_i). We define the *distance* from any class SC_c ($SC_c \succeq SC_i$) to this discontinuity as $(i+1)-c$. By convention, we consider that there is a chain discontinuity just after the last class SC_N. Let $X_t(c)$ be the distance from SC_c to the next discontinuity in the key chain at time t.

An example of the values of $X_t(c)$ is shown in Fig. 4. $X_t(c)$ is a Markov process. Indeed, $X_t(c)$ depends only on its last value and the membership change that occurs. In other words, the membership change determines in a unique way the next value of $X_t(c)$ conjointly with its previous value (and this does not depend on older history). Since there is a chain discontinuity just after SC_N, the maximum value of $X_t(c)$ is $N-c+1$, i.e., $X_t(c) \in \{1, \ldots, N-c+1\}$. For example, the maximum distance between SC_3 and the next discontinuity is $10-3+1=8$. In Fig. 4, this distance is 3 but it can change when membership changes occur.

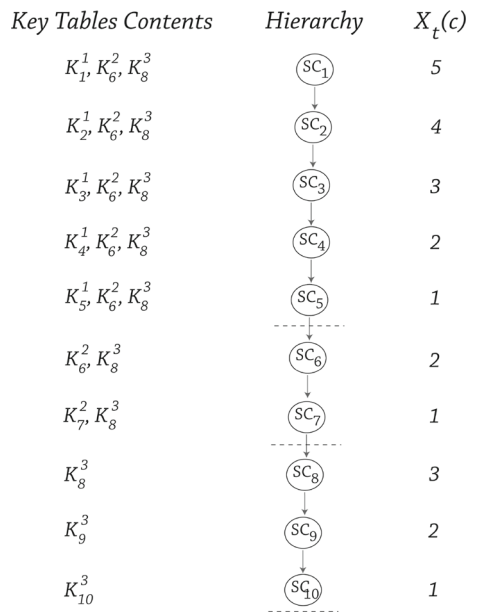

Fig. 4 Chain discontinuity process

We denote by EX_c the average value of $X_t(c)$. That is, EX_c is the average distance from SC_c to the next discontinuity. Note that EX_c does not depend on t but only depends on the class.

EX_c can be used to compute the key storage and the computation overheads. For example, let us say we find that for a hierarchy of 50 classes, $EX_c = 5$ for all values of c. This means that the average distance from the first class to the first discontinuity is five classes, then the distance from the first discontinuity to the second is five classes as well and so on. That is, the whole key chain can be divided, by the chain discontinuities, into 10 segments of five classes each. Each chain discontinuity corresponds to a key that should be stored by higher classes, so (in average) the maximum number of keys that a class should store is the number of discontinuities, which is 10 in our case (stored by users of SC_1 for example). Furthermore, because each segment contains only five classes, a user would need to apply the hash function no more than four times (5−1). As you can see in this example, knowing the value of EX_c would allow to estimate the overheads of KTLH.

We denote by $Q(c)$ the transition matrix[1] of $X_t(c)$. $Q(c)$ gives the rates at which $X_t(c)$ changes its value. In what follows, we give its analytical form for all values of c. $Q(c)$ will be used to study the existence of the limiting distribution, π^c, of $X_t(c)$.

The reason we need to study the existence of the limiting distribution π^c is that, when it exists, π^c gives the probability that $X_t(c)$ (t big enough) is in the state i. Then, the average value of $X_t(c)$, EX_c, can be obtained from π^c using [2]:

$$EX_c = \sum_{i=1}^{N-c+1} i.\pi^c(i) \tag{1}$$

and π^c in its turn can be obtained from $Q(c)$ by solving the equation system:

$$\begin{cases} \pi^c Q(c) = \pi^c \\ \sum_{i=1}^{N-c+1} \pi^c(i) = 1 \end{cases} \tag{2}$$

Allowing us to find EX_c for any class c.

In the rest of this section, we determine the transition matrix $Q(c)$ of $X_t(c)$ in 3.2; then we use $Q(c)$ in order to determine EX_c, the average value of $X_t(c)$, in 3.3.

[1] We consider transition matrices such that the sum of each row is 1.

Table 3 Used notations

Notation	Meaning
N	Number of security classes
SC_c	The *cth* security class
$X_t(c)$	The distance from SC_t to the next discontinuity at time t
EX_c	The average value of $X_t(c)$
Ω	The events probability matrix
$\omega_{x,y}$	An element of Ω
$Q(c)$	The transition matrix of $X_t(c)$
$q_{x,y}$	An element of $Q(c)$
π^c	The limiting distribution of $X_t(c)$

3.2 The transition matrix of $X_t(c)$: $Q(c)$

We denote by $\omega_{x,y}$ the probability of the event "*a user will switch from SC_x to SC_y*." By convention, SC_0 contains external hosts. That is, $\omega_{0,y}$ is the probability that an external host joins the class SC_y and $\omega_{x,0}$ is the probability that a user leaves the class SC_x and becomes an external host. By definition, $\sum_{x=0}^{N}\sum_{y=0}^{N}\omega_{x,y} = 1$. We define Ω as the $(N+1)\times(N+1)$ matrix having $\omega_{x,y}$ as entries. Ω will be used to compute $Q(c)$. Note that Ω is not a transition matrix. Table 3 summarizes the notations we will use.

$Q(c)$ is a $(N-c+1)\times(N-c+1)$ matrix because $X_t(c)\in\{1,\ldots,N-c+1\}$. Obviously, in order to obtain concrete values of π^c, we need to have concrete values of $Q(c)$ (cf. equation system 2). However, in order to stay as general as possible, we will not use numerical values for $Q(c)$, but we have rather chosen to consider the model where the membership change rate is the same for any pair of classes SC_x and SC_y; i.e., a user has the same probability to stay in their class as to switch to any other class. Our approach can still be applied in the same way to other values of $Q(c)$. In other words, all the entries of the transition matrix Ω are identical, i.e: $\forall x, y, \omega_{x,y} = \omega$, where ω is a constant. It results that $\sum_{x=0}^{N}\sum_{y=0}^{N}\omega_{x,y} = \sum_{x=0}^{N}\sum_{y=0}^{N}\omega = (N+1)^2\omega$. Since $\sum_{x=0}^{N}\sum_{y=0}^{N}\omega_{x,y} = 1$, we obtain for all x, y:

$$\omega_{x,y} = \omega = \frac{1}{(N+1)^2} \tag{3}$$

Let us denote by $q_{x,y}$ the elements of $Q(c)$. Since $Q(c)$ depends on the value of c, we distinguish three cases: $c = N$, $c = 1$ and $2 \le c \le N-1$.

(1) **c = N.**
$X_t(N)\in\{1\}$. The process $X_t(N)$ has a constant value and is permanently equal to 1. $Q(N)$ is a 1×1 matrix containing one element: $q_{1,1} = 1$.

(2) **c = 1.**
$X_t(1)\in\{1,\ldots,N\}$, $Q(N)$ is a $N\times N$ matrix. We distinguish three cases according to the values of x.
For x=1:

$$q_{1,y} = \begin{cases} \dfrac{N^2+3}{(N+1)^2}, & \textbf{if } y = 1 \\ \dfrac{2}{(N+1)^2}, & \textbf{if } 2 \le y \le N-1 \\ \dfrac{2}{(N+1)^2}, & \textbf{if } y = N \end{cases} \tag{4}$$

These formulas are based on KTLH operations. For example, $q_{1,N}$ (last line of equation system 4) gives the rates at which the distance from SC_1 to the next chain discontinuity changes from 1 to N. According to the formula above, $q_{1,N} = \frac{2}{(N+1)^2}$. This is because having a distance of N to the next chain discontinuity means that there is no chain discontinuity but the one after SC_N (by convention). This situation only happens when a user joins or leaves SC_1. A join to SC_1 occurs with the rate $\omega_{0,1} = \frac{1}{(N+1)^2}$ (i.e., from SC_0 to SC_1, SC_0 is the set of external hosts). A leave occurs with the rate $\omega_{1,0} = \frac{1}{(N+1)^2}$. Similarly, we obtain:
For x=N:

$$q_{N,y} = \begin{cases} \frac{2N}{(N+1)^2}, & \textbf{if } y = 1 \\ \frac{2(N-y+1)}{(N+1)^2}, & \textbf{if } 2 \le y \le N-1 \\ \frac{N+3}{(N+1)^2}, & \textbf{if } y = N \end{cases} \tag{5}$$

For $2 \le x \le N-1$:

$$q_{x,y} = \begin{cases} \frac{2(N-y+1)}{(N+1)^2}, & \textbf{if } 1 \le y \le x-1 \\ \frac{N+3+(N-x)(N-x+1)}{(N+1)^2}, & \textbf{if } y = x \\ \frac{2}{(N+1)^2}, & \textbf{if } x+1 \le y \le N-1 \\ \frac{2}{(N+1)^2}, & \textbf{if } y = N \end{cases} \tag{6}$$

(3) ***2 ≤ c ≤ N-1***
$X_t(c)\in\{1,\ldots,N-c+1\}$, $Q(c)$ is a $(N-c+1)\times(N-c+1)$ matrix defined by: *For x=1:*

$$q_{1,y} = \begin{cases} \frac{3N+1+(N-c-1)(N-c-4)}{(N+1)^2}, & \textbf{if } y = 1 \\ \frac{2c}{(N+1)^2}, & \textbf{if } 2 \le y \le N-c \\ \frac{c^2+c}{(N+1)^2}, & \textbf{if } y = N-c+1 \end{cases} \tag{7}$$

For x=N-c+1:

$$q_{N-c+1,y} = \begin{cases} \frac{2N}{(N+1)^2}, & \textbf{if } y = 1 \\ \frac{2(N-y+1)}{(N+1)^2}, & \textbf{if } 2 \le y \le N-c \\ \frac{N+1+c^2+c}{(N+1)^2}, & \textbf{if } y = N-c+1 \end{cases} \tag{8}$$

For $2 \leq x \leq N - c$:

$$q_{x,y} = \begin{cases} \frac{2(N-y+1)}{(N+1)^2}, & \textbf{if } 1 \leq y \leq x-1 \\ \frac{(N-c-x+1)(N-c-x+2)}{(N+1)^2} & \\ \quad + \frac{N+1+2c}{(N+1)^2}, & \textbf{if } y = x \\ \frac{2c}{(N+1)^2}, & \textbf{if } x+1 \leq y \leq N-c \\ \frac{c^2+c}{(N+1)^2}, & \textbf{if } y = N-c+1 \end{cases} \tag{9}$$

3.3 Average value of $X_t(c)$: EX_c

Using the values of $q_{x,y}$, we can give the next theorem:

Theorem 1 (Existence of the Limiting Distribution): *The limiting probability* π^c *of* $X_t(c)$ *exists [2] and is the unique solution of:*

$$\begin{cases} \pi^c . Q(c) = \pi^c \\ \sum_{i=0}^{N-c+1} \pi^c(i) = 1 \end{cases} \tag{10}$$

Proof Since $q_{i,j} > 0$ for all c, $1 \leq c \leq N$ and all $i, j \in \{1, \ldots, N-c+1\}$, any two states are communicating. Furthermore, $\{1, \ldots, N-c+1\}$ is a finite set. Therefore, according to theorem 2 in Appendix A, π^c exists and is the unique solution of the above equation system. □

We have solved, using MATLAB, equation system 10 for several values of N ranging from 2 to 1000. This allowed us to find π^c values for each value of N. Note that for each value of N there is a whole set of π^cs (a π^c for each class c between 1 and N). Then, for each value of N, we used Eq. 1 to find the values of EX_c ($1 \leq c \leq N$).

For each value of N, we distinguish three cases depending on the value of c

(1) $c = N$: $X_t(N)$ is constant and equals 1. Therefore, its average value, EX_N, equals 1.

(2) $c = 1$: According to Fig. 5, we see that EX_1 is a linear function of $log(N)$. Therefore, EX_1 is in $O(log(N))$.

(3) For c *between 2 and* $\leq N - 1$: In order to have the most reliable results, we have considered different values of N while varying c. Figures 6 and 7 show EX_c as a function of $log(N)$, for $60 \leq N \leq 200$ and $10 \leq c \leq 20$ (cf. Fig. 6), and $700 \leq N \leq 1000$ and $180 \leq c \leq 280$ (cf. Fig. 7). As for the case where $c = 1$, we observe here that EX_c has a linear shape as a function of $log(N)$.

4 Discussion

These results suggest that the average distance to the next discontinuity is of complexity $O(log(N))$ where N is the

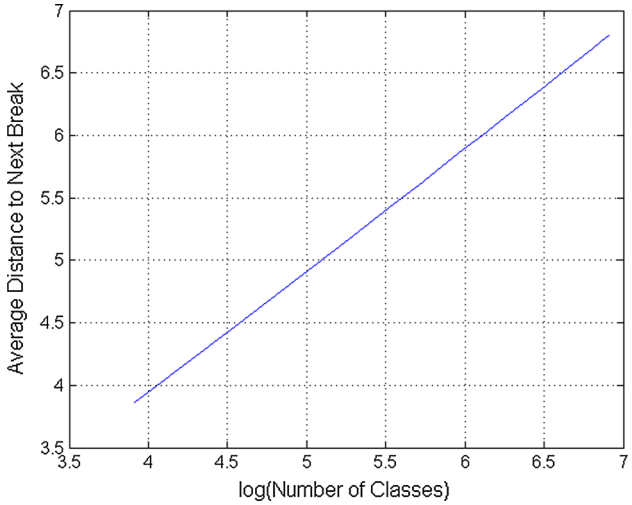

Fig. 5 EX_1 as a function of $log(N)$

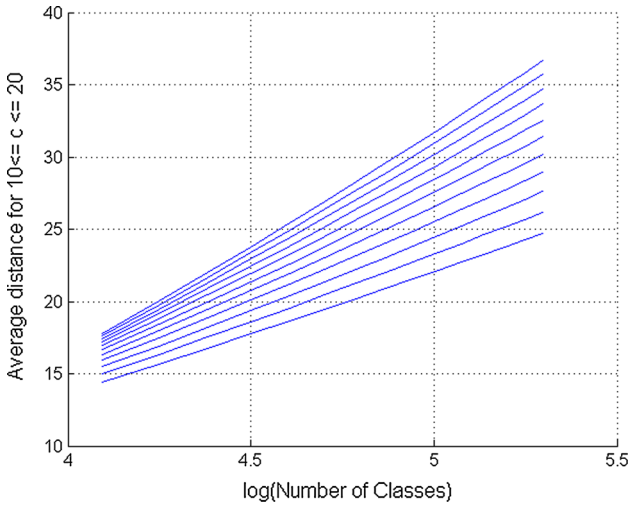

Fig. 6 EX_c as a function of $log(N)$

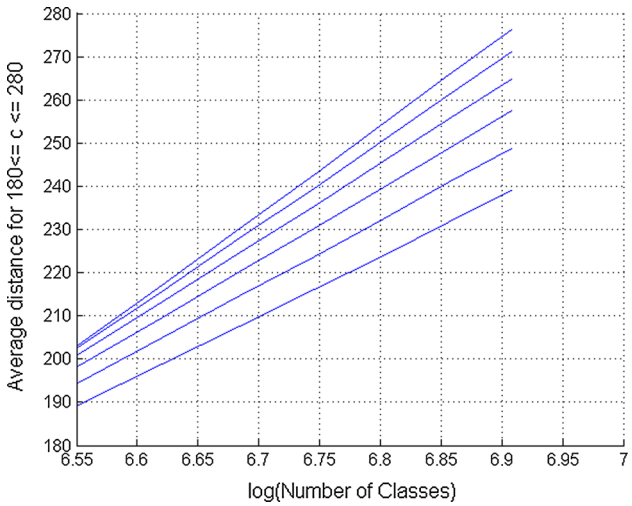

Fig. 7 EX_c as a function of $log(N)$

number of security classes. Our extensive tests confirmed this hypothesis.

4.1 Evaluation of KTLH

4.1.1 Key storage overhead

Since EX_c is of complexity $O(log(N))$, the average distance from the first discontinuity to the second one is $O(log(N))$, the average distance from the second discontinuity to the third is $O(log(N))$ and so on. Therefore, the whole key chain can be divided into K segments, each segment is of $O(log(N))$ classes. That is, K is $O(\frac{N}{log(N)})$

Each chain discontinuity corresponds to a key that should be stored by higher classes, the maximum number of keys that a class should store is the number of discontinuities K. It follows that the key storage complexity is $O(\frac{N}{log(N)})$.

4.1.2 Computation overhead

Since the key chain is split into segments of $O(log(N))$ classes, the maximum number of computations a user needs to do is the length of the segment, which is $O(log(N))$. That is, the computation overhead is $O(log(N))$.

4.1.3 Bandwidth overhead

When a user leaves or joins a security class SC_c, this results in the renewal of the key of their class and all lower classes. This is done by generating new keys for their class and lower classes and sending, to each class, its new key. Also, the new key of SC_c has to be sent to all higher classes. That is, each class in the system will need to receive exactly one message, whatever its position is in the hierarchy. The bandwidth overhead in this case is $O(1)$. On the other hand, if a user is promoted or demoted, then two chain discontinuities are created. Classes above the higher discontinuity will need to receive two new keys, whereas all other classes will need to receive only one (as shown in the example in 2.3). The number of messages that need to be sent is smaller or equal to a constant (2), and thus, the bandwidth overhead is $O(1)$ in this case as well. Table 4 shows these results.

Table 4 Performance measures of KTLH

Computation	Key storage	Bandwidth
$O(log(N))$	$O(\frac{N}{log(N)})$	$O(1)$

Table 5 A comparison of overheads of the dependent-keys approaches

Approach	Computation	Storage	Bandwidth
Indirect [9,11]	$O(N)$	$O(log(N))$	$O(N)$
Direct [12,13], [17,20]	$\simeq O((N^2))$	**O(1)**	$O(N)$
KTLH	**O(log(N))**	$O(\frac{N}{log(N)})$	**O(1)**

4.2 Comparison to the existing schemes

KTLH is a dependent-keys key management scheme. The complexity of some of the existing dependent-keys key management schemes [9,11–13,17,20] when applied to linear hierarchies was evaluated in [6]. Table 5 uses this evaluation and compares the complexity of KTLH to the dependent-keys schemes, namely direct and indirect schemes. Optimal overheads are highlighted.

KTLH has the best computation overhead, and it is the only scheme that gives an overhead that is less than the linear overhead ($O(N)$). It also has the lowest bandwidth overhead. On the other hand, The indirect and direct approaches have a better storage overhead. The fact of staying below the $O(N)$ bound for all overheads gives KTLH some advantage over the direct and indirect approaches, particularly if used for devices with limited capacity or within hierarchies with big numbers of classes.

4.3 Generalization to other key management schemes

Even if our transition matrix is specific to KTLH, the reasoning and methodological steps we presented in this paper can be applied to other key management schemes.

Finding the matrix Ω (that we defined to contain the switching probabilities from a class x to a class y) will probably have to be the first step and the corner stone in any application of Markov processes to the evaluation of a key management scheme.

Furthermore, the technique we used to solve equation system (10) and find EX_c can be easily used by other key management schemes evaluation works when faced by complex equation systems.

5 Conclusion

In this paper, we provided a theoretical evaluation of the performance of KTLH. We used the Markov processes in order to model the distance to next discontinuity from each class. We demonstrated that the average distance to the next

discontinuity is of $O(log(N))$, where N is the total number of classes. Using this result, we concluded that, for KTLH, the average key storage per user is $O(\frac{N}{log(N)})$, the average computation overhead is $O(log(N))$, and the average bandwidth overhead per key renewal is $O(1)$. Comparison to other dependent-keys schemes showed that KTLH tunes storage, computation and bandwidth overheads to achieve good performance trade-offs. Future works can extend our Markovian approach to evaluate other key management schemes as explained in the previous section.

Appendix

The Markov property

A *stochastic process* $\{X_t, t \geq 0\}$ is a family of random variables X_t where t is an indexing parameter, t is usually time. The set of values (states) in which X_t takes its values is called the *state space* S.

Let $\{X_t, t \geq 0\}$ be a stochastic process. The Markov property states that the conditional distribution of X_t at time $t+s$ $(s, t \geq 0)$ given the history of the process up to and including time t depends only on the state of the process at time t. In other words, the state of the process at time $t+s$ is conditionally independent of the history of the process before time t, given the state of the process at time t. We give hereafter a formal definition of Markov processes.

Definition 1 *(Markov Process):* A stochastic process $\{X(t), t \geq 0\}$ is a Markov process if for all $s, t \geq 0$ and nonnegative integers $i, j, x_u, 0 \leq u < s$:

$$P\{X_{s+t} = j | X_s = i, X_u = x_u, 0 \leq u < s\} = P\{X_{s+t} = j | X_s = i\} \quad (11)$$

Definition 2 *(State Distribution Vector):* Let X_t be a Markov process defined on the state space $S = \{1, \ldots, N\}$. The *state distribution vector* π_t is defined by $\pi_t = (\pi_t(1), \ldots, \pi_t(N))$ where $\pi_t(i) = P\{X_t = i\}$ for all $i \in S$.

The state distribution vector π_t defines for each time t the probability distribution of X_t.

The transition matrix

The entries of the transition (rate) matrix Q give at which rate the process moves from a state to another. Entries of Q are defined by:

$$q_{i,j} = \lim_{h \to 0} \frac{P\{X_{t+h} = j | X_t = i\}}{h} \quad (12)$$

The sum of each row of this matrix is equal to 1. If $P\{X_{t+s} = j | X_t = i\}$ does not depend on t, then X_t is said to be *homogenous*. In this case, we denote by $p_{i,j}(s)$ the probability $P\{X_{t+s} = j | X_t = i\}$.

Limiting distribution

Definition 3 *(Communicating states and irreducibility* [4]*):* Two states $i, j \in S$ are said to *communicate* if there are $t_1, t_2 > 0$ such that: $p_{i,j}(t_1) > 0$ and $p_{j,i}(t_2) > 0$. The Markov process is *irreducible* if any two states in S can communicate.

Definition 4 *(Stationary distribution vector):* The *stationary distribution vector* π verifies:

$$\pi.Q = \pi \quad (13)$$

and the normalization condition:

$$\sum_i \pi(i \in S) = 1 \quad (14)$$

Definition 5 *(Limiting distribution vector):* The limiting distribution vector $\pi*$ is defined by:

$$\pi* = \lim_{t \to \infty} \pi_t \quad (15)$$

when this limit exists

Theorem 2 (Existence of the limiting distribution [4]): *The stationary distribution vector* π *of an* irreducible *Markov process on an* finite *state space is* unique. π *is also the limiting distribution.*

That is, the limiting distribution π of an irreducible Markov process on a finite set S exists and can be computed using (13) and (14):

$$\pi.Q = \pi$$
$$\sum_{i \in S} \pi(i) = 1$$

References

1. Hassen, H.R., Bouabdallah, A., Bettahar, H., Challal, Y.: Key management for content access control in a hierarchy. Comput. Netw. **51**(11), 3197–3219 (2007). doi:10.1016/j.comnet.2006.12.011
2. Norris, J.R.: Markov Chains. Cambridge University Press, Cambridge (1999)
3. Sharpe, M.: General Theory of Markov Processes, vol. 133. Academic, Cambridge (1988)
4. Kijima, M.: Markov Processes for Stochastic Modeling. Chapman and Hall, London (1993)
5. Davis, M.H.A.: Markov Models and Optimization. Chapman and Hall, London (1993)
6. Hassen, H.R., Bettahar, H., Bouadbdallah, A., Challal, Y.: An efficient key management scheme for content access control for linear hierarchies. Comput. Netw. **56**(8), 2107–2118 (2012). doi:10.1016/j.comnet.2012.02.006
7. Wong, C.K., Gouda, M., Lam, S.S.: Secure Group Communications Using Key Graphs. ACM SIGCOMM (1998)
8. Sandhu, R.: On some cryptographic solutions for access control in a tree hierarchy. IEEE, pp. 405–410 (1987)
9. Sandhu, R.: Cryptographic implementation of a tree hierarchy for access control. Inf. Process. Lett. **27**(2), 95–98 (1988)
10. Gudes, E.: The design of a cryptography based secure file system. IEEE Trans. Softw. Eng. SE-6 **5**, 411–420 (1980)
11. Yang, C., Li, C.: Access control in a hierarchy using one-way functions. Elseveir Comput. Secur. **23**, 659–664 (2004)
12. Akl, S.G., Taylor, P.D.: Cryptographic solution to a problem of access control in a hierarchy. ACM Trans. Comput. Syst. **1**(3), 239–248 (1983)
13. Ray, I., Ray, I., Narasimhamurthi, N.: A cryptographic solution to implement access control in a hierarchy and more. In: SACMAT'02, pp. 65–73 (2002)
14. Zou, X., Ramamurthy, B., Magliveras, S.S.: Chinese remainder theorem based hierarchical access control for secure group communication. In: ICICS '01: Proceedings of the Third International Conference on Information and Communications Security, pp. 381–385 (2001)
15. Hardjono, T., Zheng, Y., Seberry, J.: New solutions to the problem of access control in a hierarchy. Tech. Rep. Preprint 93-2
16. Zheng, Hardjono, Pieprzyk: Sibling intractable function families and their applications. In: ASIACRYPT: Advances in Cryptology—ASIACRYPT: International Conference on the Theory and Application of Cryptology. LNCS, Springer (1991). [Online]. Available: citeseer.csail.mit.edu/zheng92sibling.html
17. Shen, V., Chen, T.: A novel key management scheme based on discrete logarithms and polynomial interpolations. Comput. Secur. **21**(2), 164–171 (2002)
18. Zhang, J., Li, J., Liu, X.: An efficient key management scheme for access control in a user hierarchy. Int. Conf. Inf. Technol. Comput. Sci. **1**, 550–553 (2009)
19. Tzeng, S.-F., Lee, C.-C., Lin, T.-C.: A novel key management scheme for dynamic access control in a hierarchy. Int. J. Netw. Secur. **12**(3), 178–180 (2011)
20. Das, M.L., Saxena, A., Gulati, V.P., Phatak, D.B.: Hierarchical key management scheme using polynomial interpolation. SIGOPS Oper. Syst. Rev. **39**(1), 40–47 (2005)
21. Lawler, G.F.: Introduction to Stochastic Processes. Chapman and Hall, London (1995)